# *Peta-Scale Lattice Quantum Chromodynamics on a Blue Gene/Q Supercomputer*

Jun Doi
IBM Research – Tokyo
5-6-52, Toyosu, Koto-ku,
Tokyo, Japan
doichan@jp.ibm.com

***Abstract*—Lattice Quantum Chromodynamics (QCD) is one of the most challenging applications running on massively parallel supercomputers. To reproduce these physical phenomena on a supercomputer, a precise simulation is demanded requiring well optimized and scalable code. We have optimized lattice QCD programs on Blue Gene family supercomputers and shown the strength in lattice QCD simulation. Here we optimized on the third generation Blue Gene/Q supercomputer; i) by changing the data layout, ii) by exploiting new SIMD instruction sets, and iii) by pipelining boundary data exchange to overlap communication and calculation. The optimized lattice QCD program shows excellent weak scalability on the large scale Blue Gene/Q system, and with 16 racks we sustained 1.08 Pflop/s, 32.1% of the theoretical peak performance, including the conjugate gradient solver routines.**

***Index Terms*—Lattice QCD, Blue Gene, SIMD**

## I. INTRODUCTION

Lattice Quantum Chromodynamics (QCD) is the discretized model of a theory of strong force among quarks for computer simulation. As the computational power of supercomputers increases, the scale of the feasible lattice QCD simulation also increases and the simulations can more precisely and accurately reproduce the physical phenomena on the computer. For example, the Chiral symmetry breaking [1] and the nuclear force of the Yukawa theory [2] were reproduced by lattice QCD simulations on the supercomputers.

Lattice QCD requires quite high memory and network bandwidth, and also high computational power. It is a grand challenge for computers, and so it is no exaggeration to say that lattice QCD has been accelerating the evolution of supercomputers. As proof, many supercomputers and architectures were developed explicitly for lattice QCD simulation and some of the supercomputers have been much influenced on their architectures. Examples include QCDPAX [3], QCDSP [4], QCDOC [5][6], and QPACE [7], which were especially designed to optimize the computations and the communications of the lattice QCD simulations. The Blue Gene/L supercomputer [8] is an example of a general purpose supercomputer that was influenced by the requirements of lattice QCD calculations. Blue Gene/L was developed based on the design concepts and the architecture of QCDOC.

We have been optimizing the lattice QCD simulation for Blue Gene/L and Blue Gene/P supercomputers. On Blue Gene/L, Vranas et al. reported 12.2 Tflop/s of sustained performance with 32K CPU cores [9], while on Blue Gene/P, we reported 1.88 Tflop/s of sustained performance with 4K CPU cores [10]. Now the third generation of the Blue Gene supercomputer, Blue Gene/Q [11] offers Pflops performance. Blue Gene/Q enhances the design concepts of Blue Gene/P for improved speed, power efficiency, scalability and productivity. For lattice QCD optimization, Blue Gene/Q is a very powerful and efficient computer because it has SIMD instructions for complex numbers, a high bandwidth torus network and well balanced memory hierarchy. There are major technology updates from Blue Gene/P to specifically optimize lattice QCD for a new hardware. In this paper, we describe techniques to optimize lattice QCD by exploiting new SIMD instructions and torus networks.

In section II, we briefly introduce the Blue Gene/Q supercomputer. In section III, we describe lattice QCD and the configuration of our simulation. Then we present the optimization techniques for lattice QCD simulation in section IV. Section V shows the performance and scaling results of our lattice QCD program running on Blue Gene/Q. Finally we summarize our results and discuss future works in section VI.

## II. BLUE GENE/Q OVERVIEW

Blue Gene/Q is the third generation of the Blue Gene supercomputer series, designed to scaled up to 20 Pflops or more. As with previous Blue Gene machines, Blue Gene/Q has compute nodes and IO nodes, both have the same compute chip that integrates processor cores, cache memory, memory controller, and network logic in a System-on-Chip (SoC) design [12]. There are 17 embedded PowerPC A2 processor cores in each compute chip and application programs can use 16 of those cores. Each processor core supports 4-way simultaneous multi-threading (SMT), so applications can exploit up to 64 shared memory parallel (SMP) threads per compute chip. Each processor core has a 4-way SIMD floating point unit called Quad FPU and supports fused multiply-add (FMA) operations, and thus each processor core can calculates

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00 ©2012 IEEE

8 floating point operations per clock cycle. Each Quad FPU also supports 2-way SIMD complex number arithmetic instructions. The frequency of the A2 processor core in Blue Gene/Q is 1.6 GHz, which means the theoretical peak performance per compute chip is 204.8 Gflops.

Each compute node is bidirectionally interconnected to 10 neighboring compute nodes in a 5D torus network [13]. The network logic and message unit are integrated on the compute chip and they are highly parallelized and optimized to handle multiple messages through the 10 bidirectional links, from and to 16 different processor cores. Blue Gene/Q supports remote direct memory access (DMA) functionality that allows asynchronous communication without any computational load on the processor cores. The smallest block of the 5D torus network is formed by 2x2x2x2x2 5D torus (= 32 compute nodes), and this forms a single node board. The dimensions are labeled as A, B, C, D, and E. The length of the E dimension is always 2 and always torus connected. For other dimensions, when the length is equal to or larger than 4, that dimension is torus connected, but otherwise it is mesh connected. 16 node boards form a 4x4x4x4x2 midplane (= 512 compute nodes), and one rack of Blue Gene/Q is consists of 2 midplanes (=1,024 compute nodes).

## III. LATTICE QCD

Lattice QCD is a common technique to calculate and simulate a field theory of QCD on the computer. Lattice QCD simulates the strong interaction of quarks and gluons in 3D space and time field. In lattice QCD, the quark fields are placed on the sites of 4D lattice and the gauge fields are placed as the links of the lattice sites to represent the effect of the gluon as the transporters of the quark fields. The simulation of the lattice QCD uses a kind of a finite difference method to solve the interactions. For a precise simulation to reproduce the physical phenomena, a larger lattice is desired, which requires more computational power.

The most computation and communicataion intensive procedure in lattice QCD is the Wilson-Dirac operator. The Wilson-Dirac operator is used to calculate the physical exchange between 4D lattice sites through the effects of the gluon fields in the iterative solvers using such methods as the conjugate gradient (CG) method. The Wilson-Dirac operator D is defined by Equation 1.

$$D(n) = \delta(n) - \kappa \sum_{\mu=1}^{4} \{(1-\gamma_\mu) U_\mu(n) \delta(n+\mu) + (1+\gamma_\mu) U_\mu^t(n-\mu) \delta(n-\mu)\} \quad (1)$$

In Equation 1, $\delta(n)$ is a quark field that has 12 physical values on each lattice site, $U_\mu(n)$ is a gauge field with 3x3 matrix representing the gluon effect between two lattice sites, $\gamma_\mu$ is the 4x4 projector gamma matrix of Equation 2, $\mu$ is one of 1, 2, 3 or 4 corresponding to the X, Y, Z and T dimensions. The $(n \pm \mu)$ indices the neighboring site of the $\pm\mu$ directions.

$$\gamma_1 = \begin{pmatrix} 0 & 0 & 0 & -i \\ 0 & 0 & -i & 0 \\ 0 & i & 0 & 0 \\ i & 0 & 0 & 0 \end{pmatrix} \quad \gamma_2 = \begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix}$$

$$\gamma_3 = \begin{pmatrix} 0 & 0 & -i & 0 \\ 0 & 0 & 0 & i \\ i & 0 & 0 & 0 \\ 0 & -i & 0 & 0 \end{pmatrix} \quad \gamma_4 = \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix} \quad (2)$$

The Wilson-Dirac operator can be rewritten as Equation 3 where *d* is the Wilson-Dslash term.

$$D(n) = \delta(n) - \kappa \cdot d(n) \quad (3)$$

Even-odd preconditioning is used to make the CG method converge faster. In the even-odd preconditioning, lattice sites are separated into even sites and odd sites and calculated independently. Thus the Wilson-Dslash is performed in 2 phases; *d_eo* from even sites to odd sites, then *d_oe* from odd sites to even sites as shown in Equation 4.

$$D(n) = \delta(n) - \kappa^2 \cdot d_oe(n) \cdot d_eo(n) \quad (4)$$

## IV. OPTIMIZING LATTICE QCD ON BLUE GENE/Q

### A. Consideration of Data Layouts

It is important to determine the proper data layout to store the values in quark fields and gauge fields in memory by considering optimization of the SIMD instructions and memory access.

Two basic data layouts are generally used to store the physical values in the arrays; a Structure of Arrays (SoA) or an Array of Structures (AoS). The SoA layout has separate arrays for each physical element so the computation for each element can be independently unrolled, pipelined, or SIMDized. The SoA layout is widely used for SIMD computers such as GPUs or vector computers. The AoS layout has all of the physical values grouped together as a data structure at the innermost of the array. The AoS layout is good for scalar computers to exploit cache memory because once one of the elements of the data structure is accessed, the other elements on the same cache line are also in the cache memory. Blue Gene/Q has SIMD instruction sets and it is also a scalar computer, so both SoA and AoS are candidates for the data layout.

#### 1) Structure of Arrays

A SoA layout has advantage to generate SIMD instructions for the sequential computations. However the performance of the SIMD calculations depends on the length of each sequence, and the efficiency increases for longer sequences.

For the calculations of Wilson-Dirac operator, there are non-sequential points around the boundaries of each dimension. Usually the quark and gauge fields are stored in X, Y, Z, T order, and in this case, the sequential length for inner dimensions, X or Y tend to be short when decomposing the lattice in a parallel environment. The length of the sequences can be increased by rearranging the array or by using an index table. In both cases, we need extra memory accesses, and for an index table we need special SIMDized load and store instructions with scatter and gather operations. Unfortunately these kinds of instructions are not available in Blue Gene/Q.

The SoA layout also has disadvantage in memory access. One reason is because there are 21 separate data streams: 12 for the quark fields and 9 for the gauge fields for Wilson-Dirac

operator. We have highly optimized automatic prefetch engine in Blue Gene/Q but the 21 data streams are too many to handle in the prefetch engine. The use of the cache memory is also a problem. As noted, there are non-sequential points around the boundaries of the inner dimension. To process the boundary parts of the innermost dimension, only one element is needed in the cache line and others are discarded. For Blue Gene/Q, the size of the L1 cache line is 64 bytes, so for double precision complex elements (16 bytes), only a quarter of the memory bandwidth is effectively used for boundary data handling.

*2) Array of Structures*

The AoS layout has advantages for cache memory use in hierarchical memory systems. However an AoS layout is not always SIMD friendly depending on the number of elements in the structure. In most cases, the number of elements has to be multiple of the vector width of SIMD processor to load data into the vector registers. Also there is a limitation on the data alignment for some processors when using SIMDized load and store instructions.

The AoS layout has a disadvantage in portability because the SIMD code has to be carefully optimized for the vector width of each SIMD instruction set. For example, Blue Gene/L and Blue Gene/P have 2-way SIMD instruction sets, but Blue Gene/Q has 4-way SIMD instruction set, and there is no compatibility between 4-way SIMD instructions and 2-way SIMD instructions. To generate portable SIMD code for lattice QCD calculations, the BAGEL assembler generation library [14] offers an elegant data layout. When the width of the SIMD is N, BAGEL packs N sites in the innermost array of the quark and gauge fields so that the SIMD instructions can be applied naturally. This data layout is also powerful for the future compatibility and portability of the programs, however the data accesses for the gauge fields per a calculation are multiplied by the vector width N.

*3) Our Data Layout for Blue Gene/Q*

We applied the AoS layout for the quark and gauge fields, but we did not use the BAGEL-style data layout. We choose fewer memory accesses over future compatibility. For both the quark and gauge fields, we used double-precision floating point numbers.

(i) Data layout for quark field

The structure of a quark field consists of 4 spinors, and each spinor has 3 colors defined as complex numbers. A quark field is usually stored as 3x4 complex matrix, but to use load and store instructions of the Quad FPU, it is preferred to store an even number of complex data elements in the innermost data structure, so we store each quark field as a 4x3 complex matrix, as shown in Figure 1. We store this structure in X, Y, Z, T order. The following FORTRAN definition shows the spinor field layout:

```
COMPLEX*16 W(4,3,NX,NY,NZ,NT)
```

For even-odd preconditioning, we divide the X dimension by 2 to separate the even sites and the odd sites.

```
COMPLEX*16 Weo(4,3,NX/2,NY,NZ,NT)
```

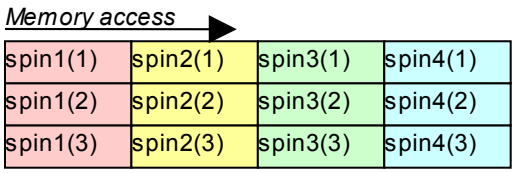


Figure 1. Data structure of 4x3 quark field

(ii) Data layout for gauge field

The structure of a gauge matrix is a 3x3 complex matrix, so the number of element of the structure is not even. However we only access 1 element at a time because we multiply the same matrix element by the vectorized spinor element. We store this structure in X, Y, Z, T order with separate gauge fields for each dimension. The following FORTRAN definition shows the gauge field layout:

```
COMPLEX*16 U(3,3,NX,NY,NZ,NT,4)
```

We also divide the X dimension for even-odd preconditioning, so we have 2 separate gauge fields for the even sites and the odd sites for each of the 4 dimensions.

```
COMPLEX*16 Ueo(3,3,NX/2,NY,NZ,NT,2,4)
```

*B. SIMDization of the Wilson-Dirac Operator*

Since we use the AoS layout, we have to SIMDize the Wilson-Dirac operator for each lattice site using Quad FPU instructions. The Quad FPU has a 2-way SIMD capability for complex number arithmetic, so we can handle 2 complex numbers in each register for each calculation step.

The pseudocode shows the Wilson-Dslash algorithm used for each lattice site:

```
Fill 4x3 quark v with zeros
For each 8 neighboring directions (X,Y,Z,T forward and backward)
    Read the 4x3 quark field w for the neighboring site
    Read the 3x3 gauge field U for the neighbor direction
    (i)Apply the gamma projector matrix to w, and the 4x3 quark w is
    integrated to the 2x3 half quark h by exploiting the symmetry of the
    gamma matrix
    (ii)Multiply U and h
    (iii)Reconstruct the 2x3 half quark h' to the 4x3 quark w' and
    accumulate in the output quark v (so v = v + k w' )
```

To calculate the Wilson-Dirac operator, we set $v$ to $w$ instead of initializing with zeros before the 8 neighbor loop, and we use the negative $k$. For the even-odd preconditioning, *d_eo* is performed using the same pseudocode, then *d_oe* is performed with the output of *d_eo* as the input $v$, and the negative $k$. In both cases, we can use the same optimized compute kernels for the Wilson-Dslash, but the memory access patterns are quite different.

The most compute intensive procedure in the Wilson-Dslash is calculating the 3x3 matrix and spinor. There are 2 spinors in half quark to be multiplied by 3x3 matrix, so this kernel can be SIMDized using 2-way complex SIMD instructions of the Quad FPU.

*1) Making Half Quark*

By exploiting the symmetry of the projector gamma matrix, we can do 4 multiplications of 3x3 matrix and spins in 2 multiplications. This technique is commonly used for lattice QCD algorithms to halve the computations and the network data transfers for the boundary exchanges.

For example, the forward direction part of the Wilson-Dslash $d_{+1}$ for X can be rewritten as shown in Equation 5,

$$d_{+1}(n) = \kappa(1-\gamma_1)U_1(n)\delta(n+1)$$
$$= \kappa\begin{pmatrix} U_1(n)\cdot(s_1+i\cdot s_4) \\ U_1(n)\cdot(s_2+i\cdot s_3) \\ U_1(n)\cdot(s_3-i\cdot s_2) \\ U_1(n)\cdot(s_4-i\cdot s_1) \end{pmatrix} = \kappa\begin{pmatrix} U_1(n)\cdot(s_1+i\cdot s_4) \\ U_1(n)\cdot(s_2+i\cdot s_3) \\ -i\cdot U_1(n)\cdot(s_2+i\cdot s_3) \\ -i\cdot U_1(n)\cdot(s_1+i\cdot s_4) \end{pmatrix} \quad (5)$$

where $\delta(n+1) = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix}$, s is the spinor.

In this case, half quark *h* is calculated as

$$h = \begin{pmatrix} s_1 + i\cdot s_4 \\ s_2 + i\cdot s_3 \end{pmatrix} \quad (6)$$

Regarding to the projector gamma matrices, the pairs of spinors (1,4) and (2,3) for X and Y dimensions, (1,3) and (2,4) for Z and T dimensions, are merged to construct half quark.

We arranged the quark spinors by column in 4x3 representation and thus we can load 2 pairs of complex elements from different spinors into the quad vector register at one time. Figure 2 shows how the spinor elements are loaded in the 6 quad vector registers.

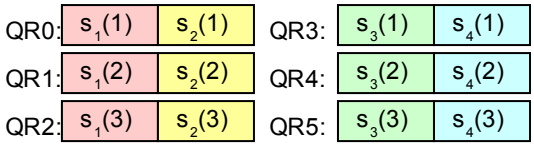



Figure 2. Loading 4x3 quark spinors in the quad vector registers

Before calculating a half quark, we have to reorder the elements from QR3 to QR5 for the X and Y dimensions. A Quad FPU has instructions to permute the elements in the quad vector registers. In this case, the QVFALIGNI instruction is used to swap 2 complex elements directly in the quad vector register. Thus we first load elements in QR3 to QR5 and then QVFALIGNI is used. The elements for QR0 to QR2 are already in the L1 cache because the size of the L1 cache line is 64 bytes and 4 sequential complex elements in QR0 and QR3 are on the same cache line. This technique is useful to optimize the pipeline scheduling for load and store instructions because loading or storing from or to L1 cache takes a constant number of cycles. We also use QVFALIGNI instruction without reordering for Z and T dimension to get other elements in the L1 cache.

Then we calculate the half quark as shown in Figure 3.

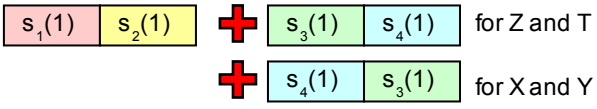



Figure 3. Making half quarks using Quad FPU

In Figure 3, "+" is the complex FMA operator varied in 8 directions. For example, the forward direction half quark for X is calculated by QR0 + i * QR3. TABLE I shows the Quad FPU instructions for each direction and constant complex values to multiply to QR3 to QR5.

TABLE I. QUAD FPU INSTRUCTIONS TO MAKE HALF QUARKS

| Axis | Multiplied constant | Forward | Backward |
|---|---|---|---|
| X | (1.0,1.0) | QVFXXNPMADD | QVFXXCPNMADD |
| Y | (1.0,-1.0) | QVFMADD | QVFNMSUB |
| Z | (1.0,-1.0) | QVFXXNPMADD | QVFXXCPNMADD |
| T | N.A. | QVFADD | QVFSUB |

*2) Multiplying gauge matrix*

Now we have 3 quad vector registers having elements of 2 pairs of spinor, each pair can be multiplied by the same element from 3x3 gauge matrix. For our usage, the Quad FPU has a complex load instruction that loads 1 complex number twice as a pair of 2 same complex numbers in the quad vector register.

To optimize the pipeline scheduling we also use the technique to load complex elements of 3x3 gauge matrix from different cache lines at first, then after multiplying these elements, other elements are already in the L1 cache. Figure 4 shows 4 contiguous 3x3 gauge matrices stored in memory and the grouping boundaries of the L1 cache lines. We load 3 diagonal elements at first because they are stored in the different cache lines in all 4 cases.

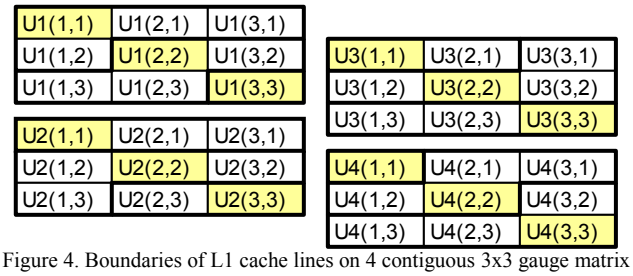



Figure 4. Boundaries of L1 cache lines on 4 contiguous 3x3 gauge matrix array

Therefore we multiply the gauge matrix in 3 steps as shown in Figure 5. First we load 3 diagonal matrix elements and multiply them and 3 pairs of elements of half quarks. For the other steps, the matrix elements can be loaded from the L1 cache memory, then we multiply and add to the output half quark by exploiting the FMA instructions.

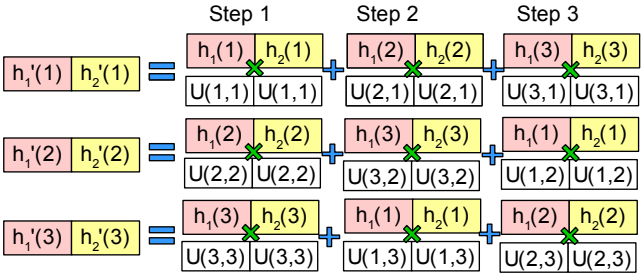



Figure 5. Multiplication of 3x3 gauge matrix and a pair of 2 spinors using Quad FPU for forward directions. For backward directions we multiply Hermitian matrix in the same manner.

A multiplication of a pair of 2 complex numbers in the quad vector register is calculated by using 2 Quad FPU

instructions as shown in Figure 6. The QVFXMUL instruction multiplies the real part of the complex *a* and the complex *b*. QVFXMADD is the FMA version of QVFXMUL instruction used for Step 2 and 3 in Figure 5. The QVFXXNPMADD instruction multiplies the imaginary part of the complex *a* and the complex *b*. Then imaginary part of the result is negated and the real and imaginary parts are swapped to add to the complex *c*. For Hermitian matrix multiplication, the QVFXXCPNMADD instruction is used to multiply the conjugate complex number instead of QVFXXNPMADD instruction.

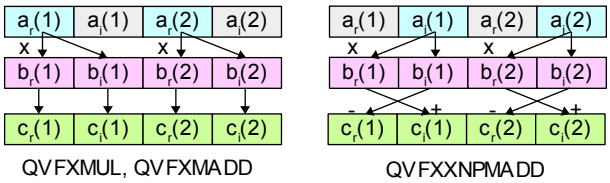


Figure 6. The Quad FPU instructions for a pair of 2 complex multiplications

*3) Accumulating quarks*

The multiplied half quark *h'* is multiplied by a scalar *k* and accumulated as the output quark as shown in Equation 5. As same as making the half quarks, we reorder the spinor elements by using QVFALIGNI instructions before accumulating spinor 3 and 4 of the output quark for X and Y dimensions as shown in Figure 7. The QVFMADD instruction accumulates spinor 1 and 2, and Quad FPU instructions shown in TABLE II are used for spinor 3 and 4.

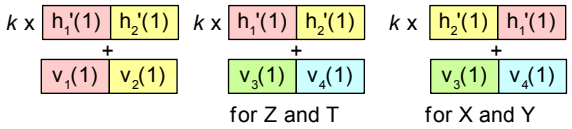


Figure 7. Accumulating half quark to output quark

TABLE II. QUAD FPU INSTRUCTIONS TO ACCUMULATE TO OUTPUT QUARK

| Axis | Multiplied constant | Forward | Backward |
|---|---|---|---|
| X | (*k*,*k*) | QVFXXCPNMADD | QVFXXNPMADD |
| Y | (*k*,-*k*) | QVFNMSUB | QVFMADD |
| Z | (*k*,-*k*) | QVFXXCPNMADD | QVFXXNPMADD |
| T | (*k*,*k*) | QVFMADD | QVFNMSUB |

*4) Optimizing instruction pipeline*

As we optimize the Wilson-Dirac operation using inline assembly, we have to carefully manage the instruction pipeline scheduling and register allocations. The following pseudocode shows part of the optimized Wilson-Dirac operator for X forward direction.

```
h1 = QVFALIGNI(w3,w3,2)
h2 = QVFALIGNI(w4,w4,2)
h3 = QVFALIGNI(w5,w5,2)
h1 = QVFXXCPNMADD(unit,h1,w1)
h2 = QVFXXCPNMADD(unit,h2,w2)
h3 = QVFXXCPNMADD(unit,h3,w3)
h1' = QVFXMUL(u11,h1)
h2' = QVFXMUL(u22,h2)
h3' = QVFXMUL(u33,h3)
h1' = QVFXXNPMADD(u11,h1,h1')
h2' = QVFXXNPMADD(u22,h2,h2')
h3' = QVFXXNPMADD(u33,h3,h3')
...
```

First 6 lines of the pseudocode make the half quark, and the next lines multiply the gauge matrix. In the actual code, we also have load and store instructions between floating operations. We can see triplets of instructions in the pseudocode, and there are only 3 cycles before the next dependent instructions use the results of the previous calculations. The latency for the complex operations of Quad FPU is 8 cycles. So the pipeline always stops at each triplet of instructions.

To avoid this pipeline stalls, we merged the procedures for the forward and backward Wilson-Dirac calculations in the same calculation loop. In this situation, there are now 6 cycles to the next dependent instructions, but there are still pipeline stalls. To avoid these stalls, exploiting 2 or more hardware threads per processor core is necessary, so that as the pipeline of one thread stalls the other threads can run. There is another major reason to exploit the hardware threads in the Blue Gene/Q core. We can execute 1 floating operation and 1 integer operation including a load or store in the same clock cycle in the Blue Gene/Q core, however we can issue only 1 operation per cycle from a thread. By exploiting 2 or more hardware threads per processor core, we can overlap floating and integer operations. This means we have to utilize at least 32 threads per compute node to maximize the performance.

*5) Performance limitation of Wilson-Dirac*

For the Wilson-Dirac calculation, in theory there are 186 floating operations in each direction. Using Quad FPU instructions to SIMDize the Wilson-Dirac operator, we have (i) 6 instructions for half quark creation, (ii) 18 instructions for matrix multiplication, (iii) 6 or 9 instructions (the 9 for X and Y) for quark accumulation, for an average of 31.5 instructions per direction. The Quad FPU has 4-way FMA capability per instruction, this means 252 floating operations is the theoretical peak performance for 31.5 instructions and the actual performance limit for the Wilson-Dirac operator is 73.8% of the theoretical peak performance. In addition, there are integer instructions to handle pointers and loops, instruction latencies and memory access latencies that further reduce the sustained performance.

## C. Parallelizing lattice QCD

A lattice QCD calculation is executed on a finite 4D lattice, so in most implementations an array decomposition is used to divide the lattice into small domains for each parallel process. We divide 4D lattice in all 4 dimensions to maximize the potential number of processes for maximum scalability. Therefore we have to exchange data among 8 neighboring processes. Blue Gene/Q has a 5D torus network and each link has bidirectional connections. The maximum network bandwidth is measured when we use all 10 connections at the same time, so it is important to communicate with as many links as possible. Therefore dividing lattice in all 4 directions is also advantageous to exploit the high communication bandwidth of Blue Gene/Q.

*1) Optimizing the boundary exchange*

In the Wilson-Dirac calculation, each site refers to the gauge fields and quark fields of the neighboring sites. Therefore boundary sites on a small domain only require the

information from the boundary sites in the neighboring small domains. The boundary data to be exchanged between neighboring processes is stored as a half quark array and its length is the 3D boundary volume, which depends on the shape of the small 4D domain. The length of the boundary array is smallest when the shape of the small domain is symmetric.

We overlap the calculations and the data exchanges to hide the communication latencies by using asynchronous communications. There are 4 independent calculations and communications related to 4 dimensions in Wilson-Dirac calculation, so we arranged these independent procedures in a 4-way pipeline to overlap the calculations and communications as shown in Figure 8.

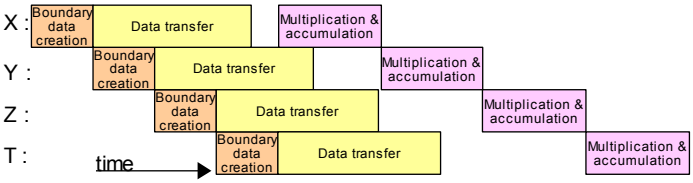



Figure 8. Overlapping calculations and communications by using pipelines for the independent procedures in the 4 dimensions

In the boundary data creation phase, we make half quarks on the boundary lattice sites. For boundary half quarks sent to the forward neighbors, we multiply the gauge matrix of forward gauge link that is referred to as a backward gauge link in the forward neighbor. We do not calculate the half quarks for the inner sites in this phase but pick them up later.

In the multiplication and accumulation phase, we multiply gauge matrices and accumulate the results in the output quark array for both the boundary and inner sites.

The data transfers are overlapped behind these 2 phases and also can be overlapped with each others transfers. We do not use MPI for asynchronous data transfers but we used the low-level SPI APIs for the DMA calls because the SPI overhead is much smaller than MPI's. Using the DMA engine allows most of the computational power of the CPU cores can be used for the calculations.

*2) Hybrid parallelization*

Each compute node of Blue Gene/Q has 64 hardware threads and there are various combination of process and thread execution mode, from a flat-MPI mode to a full SMP mode. As we discussed in IV.B.4) at least 32 threads or processes per compute node are required. We implemented the Wilson-Dirac operator using OpenMP for SMP threads and our code supports various combinations of processes per compute node and threads per processes. Figure 9 shows the measured elapsed time for the lattice QCD simulation on Blue Gene/Q. In these measurements, we changed the number of processes per compute node from 1 to 32, but we held the total number of hardware threads per node to 32. This means the number of threads per process is calculated by dividing 32 by the number of processes per compute node.

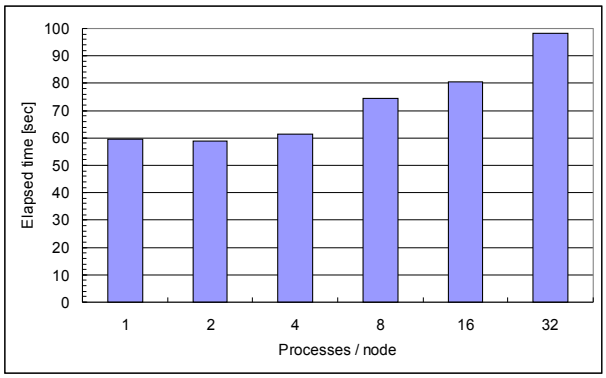



Figure 9. Comparison of the elapsed time of lattice QCD simulation with different numbers of processes per node

In these measurements, using 2 processes per compute node shows the best result and the elapsed time increases when we increase the number of processes per compute node. We choose 1 process per compute node to run our lattice QCD simulation on Blue Gene/Q because the performance was close to the result with 2 processes but we can simplify the program and the configuration. This comparisons were necessary to understand the characteristics of the program and allowed us to choose the best combination of processes and threads.

*3) Optimizing node mapping*

The Wilson-Dirac operator only exchanges boundary data between neighboring small domains allocated to parallel processes, so the network communication paths do not change and it is important to minimize the paths to reduce the communication latencies. The torus network has an advantage for these kind of communication patterns because the communication paths can be limited by carefully mapping the problem onto the torus network.

The lattice QCD is a 4D problem, however the torus network of Blue Gene/Q has a 5D torus network, so we can not directly map the 4D lattice onto the 5D torus network. We can reorder the process IDs allocated to each compute node to form a ring connected ordering in a 2D sub-torus network to effectively reduce the dimension of the torus network (Figure 10). By using this reordering, the pairs of contiguous processes are physically neighbors in the torus network. Blue Gene/Q can change the node mapping by using a mapfile at the runtime. We developed a generator to make the mapfile from 2 parameters, the 5D torus shape and 4D process counts.

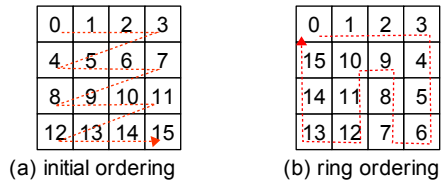



Figure 10. Example of reordering processes in a 2D 4x4 sub-torus network to form a ring connected ordering

In addition, when we run the application with multiple processes per compute node, the communication between processes in the same compute node is treated as a 6th dimension in a virtual 6D torus network. In this case, we construct 2 ring connected ordering using 2 pairs of 2D sub

-torus or 1 ring connected ordering in a 3D sub-torus to reduce the dimension of the torus network to 4D. TABLE III shows some examples of 4D node mapping on the torus network of Blue Gene/Q. We represent 5 axis of the 5D torus network as A, B, C, D and E, and the 6th dimension as T.

TABLE III. EXAMPLES OF 4D MAPPING OF TORUS NETWORK

| Configuration | Torus shape | Combination | 4D torus |
|---|---|---|---|
| 1 rack | 4x4x4x8x2 | AxBxCExD | 4x4x8x8 |
| 1 rack, 32 process per node | 4x4x4x8x2x32 | AExDxBCxT | 8x8x16x32 |
| 4 racks | 4x8x8x8x2 | AExBxCxD | 8x8x8x8 |
| 16 racks | 16x8x8x8x2 | BxCxDxAE | 8x8x8x32 |

## V. PERFORMANCE RESULTS

We created a solver program to measure the performance of Wilson-Dirac operator kernel. We used the BiCGStab method for the iterative solver routine in the program. In the iterative solver, we also hand tuned the linear algebra routines to exploit the Quad FPU. We tested 12 different initial values of spinor field and solved 12 equations to measure the performance.

We used the performance monitor library for Blue Gene/Q to count the floating point instructions to assess the sustained performance. With the performance monitor library, we can independently and simultaneously count 6 different types of floating point instructions for each hardware thread. The 6 types are; (A) add and subtract, (B) multiply, (C) FMA, (D) Quad FPU add and subtract, (E) Quad FPU multiply, and (F) Quad FPU FMA. The floating operations are totaled with Equation 7.

$$fops = A + B + 2C + 4D + 4E + 8F \quad (7)$$

We also measured the elapsed times for the solvers to calculate the sustained performance by dividing the total floating operation counts by the elapsed times.

We tested our lattice QCD program on the 16-rack Blue Gene/Q system. Our test used 1, 2, 4, 8 and 16 racks to study the scaling. The configuration of the 5D torus network on 16-rack system and the 4D mapping we used for the measurements are shown in TABLE IV.

TABLE IV. SHAPE OF 5D TORUS NETWORK AND 4D MAPPING

| Configuration | Torus shape | 4D mapping |
|---|---|---|
| 1 rack | 4x4x4x8x2 | 8x8x4x4 |
| 2 racks | 4x4x4x8x2 | 8x8x8x4 |
| 4 racks | 8x4x8x8x2 | 8x8x8x8 |
| 8 racks | 16x4x8x8x2 | 8x8x8x16 |
| 16 racks | 16x8x8x8x2 | 8x8x8x32 |

Before discussing the scalability measurements, we show the comparisons of the performance with and without using mapfile in Figure 11.

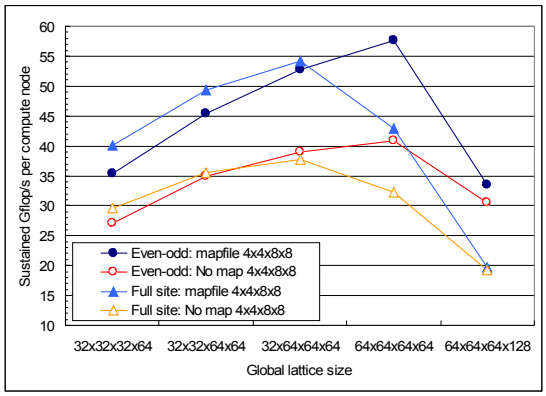



Figure 11. Comparisons of the sustained performance per compute node with and without mapfile on 1 rack of Blue Gene/Q

We measured the sustained performance of the lattice QCD calculation on 1 rack of Blue Gene/Q for 5 different sizes of lattice. We decompose the lattice in 4D, 4x4x8x8 processes and compared the sustained performance with and without using mapfile to study the effect of the ring ordering of 4D virtual torus. For 4 lattice sizes from 32x32x32x64 to 64x64x64x64, most of the calculations are done in the L2 cache, and we gained 30 – 40% better performance by using mapfile for a ring ordering. For larger lattice size, we gained slightly better performance.

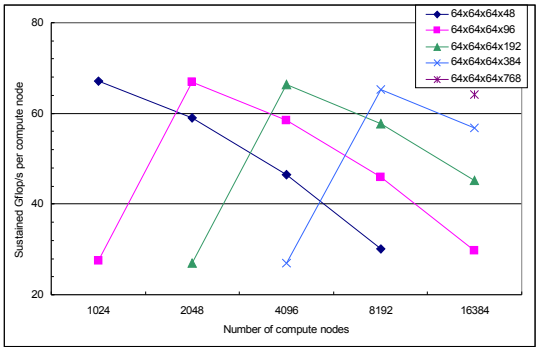
We prepared 3 types of lattice to study the scaling. Usually the sizes of the 3D spaces in the X, Y and Z dimensions in the 4D lattice are the same and the time T is varied to determine the precision of the simulation. For the weak scaling measurement, we changed the T size as shown in TABLE V. We can set the number of threads per process to 32 or 48 so that we can divide the local lattice by the number of threads.

TABLE V. LATTICE SIZE AND NUMBER OF THREADS FOR MEASUREMENTS

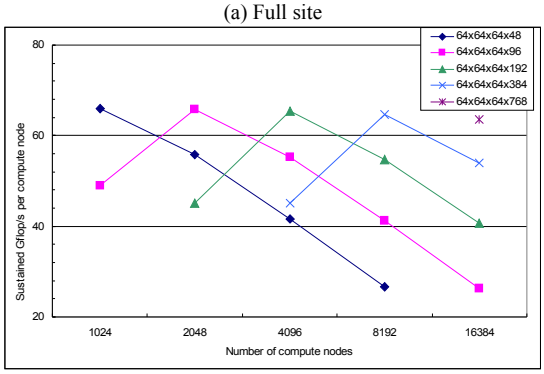
| Lattice Size (n=1, 2, 4, 8, ...) | Number of Threads |
|---|---|
| 64x64x64x12n | 48 |
| 64x64x64x16n | 32 |
| 96x96x96x12n | 48 |

### A. *Strong scaling*

By running the lattice QCD program with different numbers of nodes with the same global lattice size, we can observe the strong scaling. Figure 12, Figure 13 and Figure 14 show the comparisons of the measured sustained performance per compute node for 3 lattice sizes presented in TABLE V.

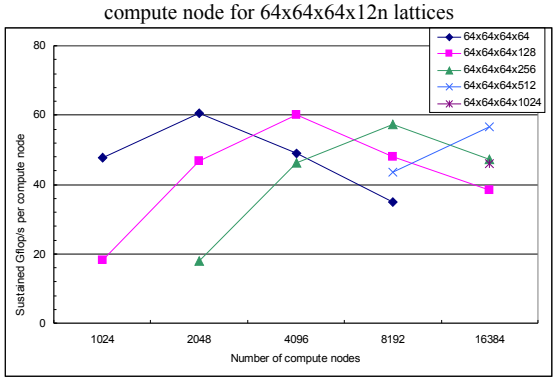



(a) Full site

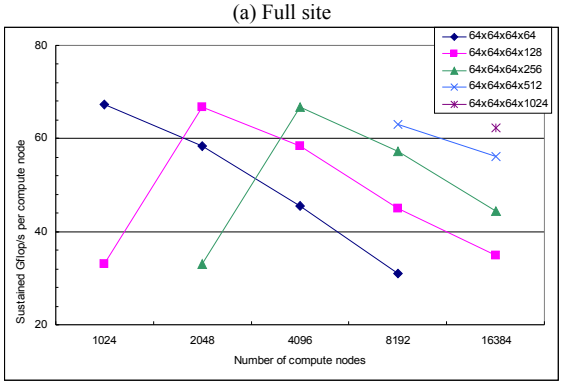



(b) Even-odd preconditionning

Figure 12. Strong scaling measurement of the sustained performance per compute node for 64x64x64x12n lattices

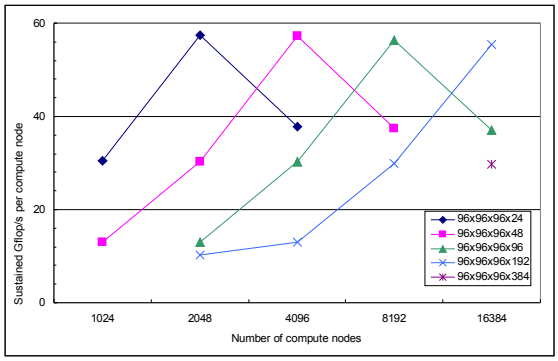



(a) Full site

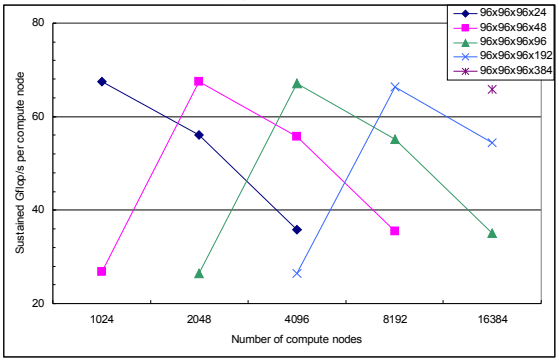



(b) Even-odd preconditionning

Figure 13. Strong scaling measurement of the sustained performance per compute node for 64x64x64x16n lattices




(a) Full site




(b) Even-odd preconditionning

Figure 14. Strong scaling measurement of the sustained performance per compute node for 96x96x96x12n lattices

Each line of these graph has a peak and the performance degrades in 2 ways. On the left side the local lattice size is too large for the L2 cache memory so the cache misses cause the performance degradation. On the right side the length of the loop in the kernel codes is too short and the resulting loop overhead causes the performance degradation. All sizes of lattice show similar results, but 48 threads per node has slightly superior performance. We also tested 64 threads per node, but the best performance was with 32 threads.

Therefore it is important to adjust the size of the local volume for each compute node to maximize the sustained performance. Figure 15 shows a performance comparison of different lattice volumes in each compute node. The local lattice size varies from 3,072 to 16,384, which is a practical range for simulations. For example, a simulation with the lattice 32x32x32x64 runs efficiently on 128, 256 or 512 nodes of Blue Gene/Q.

### *B. Weak scaling*

By fixing the local lattice size, we can study weak scaling. Figure 16 compares the sustained performance per compute node for 3 sizes of local lattice. The labels of these graphs present the local lattice size for each compute node. In Figure 16, as the number of nodes increases, the performance is slightly degraded. For the Wilson-Dirac operator, the algorithm has a potential for perfect weak scaling because the local calculations and nearest neighbor communication patterns are not influenced by the number of processes. However for the conjugate gradient solver, there are collective communications required to calculate the sum of the dot product calculated by each process. The communication latency of the collective

communication on the torus network depends on the size and shape of the torus network, but basically is proportional to the maximum length of the torus in any dimension, so a symmetric torus has better performance. Unfortunately our 16 rack system is asymmetric for 8-rack configuration of 16x4x8x8x2, which causes a large performance gap between 4-rack result and 8-rack result. However the elapsed time for the collective communications is relatively quite small in the lattice QCD calculations, so the performance degradation is relatively small and we still observed excellent scalability, as shown in Figure 17.

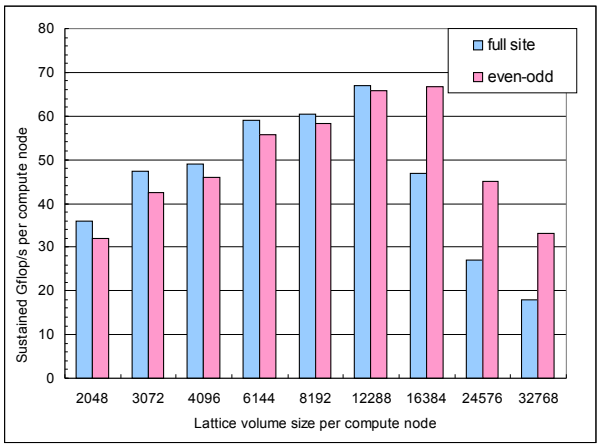
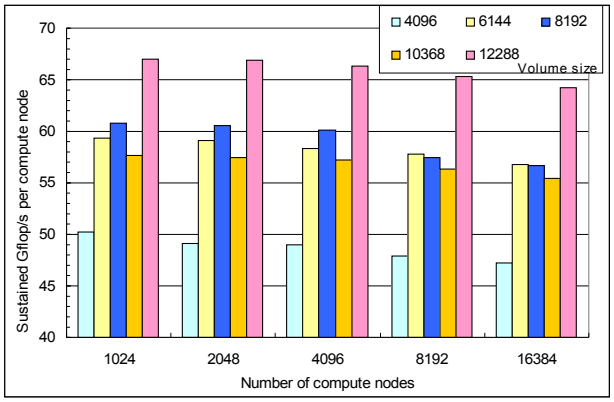
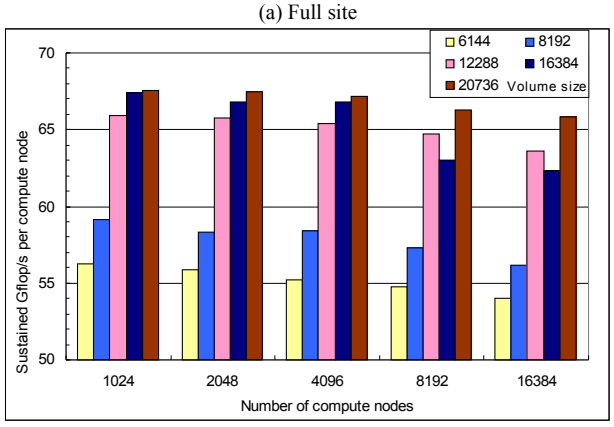



Figure 15. Comparison of the sustained performance per compute node with different size of local lattice volume sizes on 2 racks of Blue Gene/Q




(a) Full site




(b) Even-odd preconditionning

Figure 16. Comparison of the weak scaling of the sustained performance per compute node with 5 different lattice volume sizes per compute node

We sustained 1.08 Pflop/s on 16 racks of Blue Gene/Q, which is 32.1% of the theoretical peak performance. A 20-Pflops system can be constructed with 96 racks of Blue Gene/Q. If we run our lattice QCD program on the 96-rack system, we estimate that we will sustain approximately 6.1 Pflop/s.

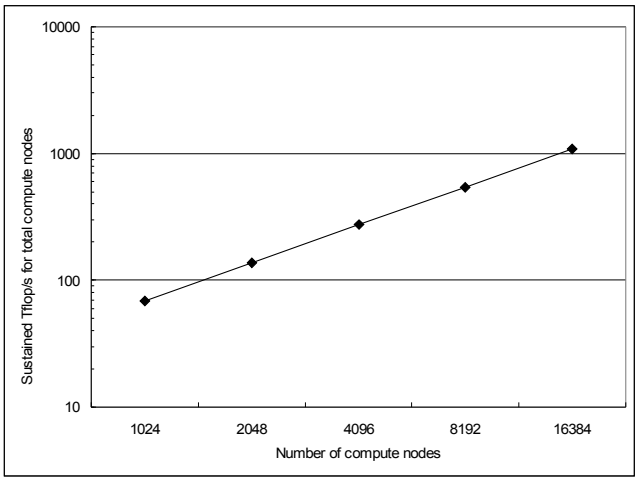



Figure 17. Scaling of the total sustained Tflop/s of lattice QCD (Even-odd preconditionning) on a Blue Gene/Q supercomputer

## VI. SUMMARY AND FUTURE WORKS

In this paper, we summarized the optimization techniques for the lattice QCD application on a Blue Gene/Q supercomputer. We presented the optimal data layouts, SIMDization techniques, how to overlap calculations and communications, and optimal node mapping for Wilson-Dirac operator on the 5D torus network. We tested our lattice QCD program on a 16-rack Blue Gene/Q system, which showed excellent sustained performance of over 1 Pflop/s, equivalent to 32% of the theoretical peak performance. We believe our optimized code will scale well for even larger systems.

We did not discuss the cache optimization in this paper. To maximize the performance and to enhance the practical local lattice size, cache optimization technique is required to reduce cache misses. Smelyanskiy et al. reported on elegant cache optimization techniques for the Wilson-Dslash [15] especially for even-odd preconditioning. We are also planning to optimize the cache usage in our lattice QCD codes in the future.

We are also working on ways to enhance our lattice QCD kernel code for 5D algorithms such as Domain-Wall. We will need larger local volumes for these 5D kernels, so we also need cache optimization for this approach.

Through these optimization works of the lattice QCD kernels, we are providing and implementing the optimal lattice QCD kernel codes in the actual lattice QCD simulation programs for the Blue Gene/Q customers. The actual simulations have been just started, and we estimate we will show some physical results of the lattice QCD simulation in several months or years.

As supercomputers continue to improve, we will continue our work to optimize lattice QCD codes for the next

generations of supercomputers that will go beyond exa-scale computing.

## Acknowledgments

We developed and tested our code on Blue Gene/Q systems at IBM Rochester and the IBM Research – T. J. Watson Research Center. We would like to thank all of the Blue Gene project members, especially Steven Westerbeck for support for the 16-rack system, James Sexton and Takao Moriyama for managerial support for our research, and Shoji Hashimoto and Hideo Matsufuru from KEK, the High Energy Accelerator Research Organization, who provided advice about Lattice QCD algorithms.

## References

[1] H. Fukaya et al. [JLQCD collaboration], Two-flavor lattice QCD simulation in the epsilon-regime with exact chiral symmetry, Physical Review Letters 98, 172001, 2007.

[2] N. Ishii et al. Nuclear force from lattice QCD, Physical Review Letters, June, 2007.

[3] T. Shirakawa et al. QCDPAX–an MIMD array vector processors for the numerical simulation of quantum chromodynamics, Proceedings of the 1989 ACM/IEEE conference on Supercomputing, 1989.

[4] R. D. Mawhinney, The 1 Teraflops QCDSP Computer, Parallel Computer 25, No. 10/11, pp.1281-1296, September, 1999.

[5] P. A. Boyle et al. QCDOC: A 10–Teraflops Computer for Tightly-Coupled Calculations, Proceedings of the ACM/IEEE conference on Supercomputing SC04, 2004.

[6] P. A. Boyle et al. Overview of the QCDSP and QCDOC computers, IBM J. of Research and Development Vol. 49, No.2/3, pp.351-365, 2005.

[7] H. Baier et al. QPACE – a QCD parallel computer based on Cell processors, PoS 27th International Symposium on Lattice Field Theory, Lattice2009, 2009.

[8] A. Gara et al. Overview of the Blue Gene/L System Architecture, IBM J. of Research and Development Vol. 49, No. 2/3, pp. 195-212, 2005.

[9] P. Vranas et al. The Blue Gene/L Supercomputer and Quantum ChromoDynamics, Supercomputing '06 Proceedings of the ACM/IEEE conference on Supercomputing SC06, 2006.

[10] J. Doi, Performance evaluation and tuning of lattice QCD on the next generation Blue Gene, PoS 25th International Symposium on Lattice Field Theory, Lattice2007, 2007.

[11] Introduction to Blue Gene/Q, http://www.ibm.com/common/ssi/cgi-bin/ssialias?subtype=WH&infotype=SA&appname=STGE_DC_DC_USEN&htmlfid=DCL12345USEN&attachment=DCL12345USEN.PDF.

[12] The IBM Blue Gene Team. The Blue Gene/Q Compute Chip, Presented at the Hot Chips Conference 23, August 17-19, 2011.

[13] D. Chen et al. The IBM Blue Gene/Q Interconnection Network and Message Unit, Supercomputing '11 Proceedings of the ACM/IEEE conference on Supercomputing SC11, 2011.

[14] P. A. Boyle, The BAGEL assembler generation library, In Proceedings of Computer Physics Communications, 2009, pp.2739-2748.

[15] M. Smelyanskiy et al. High-Performance Lattice QCD for Multi-core Based Parallel Systems Using a Cache-Friendly Hybrid Threaded-MPI Approach, Supercomputing '11 Proceedings of the ACM/IEEE conference on Supercomputing SC11, 2011.